‘완전한 계산 기계’의 꿈으로 탄생한 컴퓨터, 모든 문제를 올바르게 해결할 수 없다

박혜린

2022. 9. 12.

Abstract

1936년, 튜링 기계(Turing machine)는 당시 수학자였던 앨런 튜링의 ‘완전한 계산 기계’라는 목표로 만들어진 현대 컴퓨터의 근간이다. 하지만 앨런 튜링은 튜링 기계를 만들고, 자신이 목표로 한 ‘완전한 계산 기계’는 만들 수 없음을 밝혀냈다. 즉, 컴퓨터는 간단한 계산을 조합한 알고리즘을 주면 알아서 그 계산들을 실행해 답을 내주는데, 모든 문제에 대해 풀 수 있는 알고리즘을 항상 찾을 수는 없다는 것이다. 이로써 튜링은 수학의 결정 불가능성(undecidability)을 보였다. 더불어 주어지는 알고리즘이 올바른지 확인하는 것 또한 쉽지 않다. 가능한 모든 입력값들에 대해 올바른 답을 내놓는 것을 확인해야하는데 가능한 입력값들의 경우의 수가 우리가 확인할 수 있는 개수보다 항상 더 많을 수 있기 때문이다. 이렇게 비록 모든 문제를 해결할 수도, 실행의 정확성을 보장할 수도 없는 컴퓨터이지만, 그렇다고 현실에서 무용지물인 것은 아니다. 현실에서의 쓰임과 알고리즘의 성질에 대한 정보를 잘 활용하면 적당히 올바른 답을 내주는 컴퓨터를 작동시킬 수 있다.

오늘날 우리가 부르는 ‘컴퓨터’는 1936년 앨런 튜링이 ‘완전한 계산 기계’로 고안한 튜링 기계(Turing machine)에서 시작되었다. 여기서 계산은 가장 간단한 연산인 사칙연산을 말하며, 이들의 조합이 곧 알고리즘이다. 따라서 컴퓨터로 어떤 문제를 해결하고 싶다면, 그 문제를 풀 수 있는 알고리즘으로 프로그램을 만들어 실행시키면 된다. 그렇다면 ‘완전한 계산’이라는 것을 어떻게 정의해야 컴퓨터가 우리가 원하는 모든 계산을 알아서 해주는 기계라고 말할 수 있을까? 우선 계산의 결과로 틀린 답이 나오지 않아야 할 것이다. 즉, 주어진 알고리즘이 정확하게 실행되어야 한다. 또한, 어떤 문제가 주어지든 연산을 잘 조합한 알고리즘을 실행해 이를 해결할 수 있으면 완전하다고 할 수 있을 것이다. 종합해 앞선 질문을 다시 적어보면, ‘모든 문제에 대한 실행 가능한 옳은 알고리즘이 존재하는가?’로 바꿀 수 있다.

‘완전한 계산 기계’라는 꿈으로 탄생한 컴퓨터를 동작시키는 건 소프트웨어(software)이다. 이 소프트웨어가 실행하는 알고리즘의 문제해결력과 정확성에 대해 살펴봄으로써, 컴퓨터에 대한 이해를 높이고 더 잘 활용할 수 있을 것이다.

소프트웨어는 원하는 문제를 항상 해결할 수 없다는 근본적인 한계점을 갖는다. 이 한계점의 발견이 사실은 컴퓨터의 시작이었다. 1900년대 초, 힐베르트가 가졌던 완벽한 수학 체계를 향한 꿈 중에는 수학의 결정가능성(decidablity)이 있었다. 이는 어떤 명제가 주어졌을 때 이 명제가 자명한 공리를 따르는지 여부를 결정할 수 있는 알고리즘이 존재하는 성질을 말한다. 수학의 결정가능성을 확인하기 위해 튜링은 튜링 기계를 만들었던 것이었지만, 튜링 기계가 풀 수 없는 문제를 찾아버렸다. (정지 문제(Halting problem)가 이에 해당한다.) 따라서 결론적으로 튜링은 수학의 결정불가능성(undecidability)을 밝혀냈다. 컴퓨터가 계산을 하는 기계하는 점에서 수학에서 시작한 것으로 볼 수 있는데, 그렇기 때문에 항상 문제를 해결하는 알고리즘이 존재하지는 않는다는 한계점은 수학의 특성을 그대로 물려 받은 것으로 자명한 것으로 보인다. 튜링은 정지 문제를 결정가능성의 반례를 찾았지만, 간단히 결정 문제(decision problem, 예 또는 아니오가 답인 문제)만 고려해봐도 모든 가능한 알고리즘으로 모든 결정 문제를 해결할 수 없음을 보일 수 있다. 우선, 괴델의 불완전성 정리에서 사용된 괴델수의 개념처럼, 모든 글자와 기호를 비롯한 모든 표기를 자연수로 나타낼 수 있다. 일단 자연수의 개수를 $\aleph$개로 표기해보자. 알고리즘도 계산의 조합으로 숫자와 연산 기호의 나열이므로 각 알고리즘은 자연수로 나타낼 수 있으며, 다른 알고리즘은 다른 자연수로 표현될 것이다. 즉, 우리가 만들어낼 수 있는 알고리즘의 수는 $\aleph$개이다. 결정 문제의 가능한 입력값 또한 $\aleph$개이고 답은 예 또는 아니오이기 때문에 결정 문제의 답이 ‘예’가 되는 입력값의 집합의 경우의 수는 $2^{\aleph}$개이다. 따라서 서로 다른 알고리즘이 $2^{\aleph}$개 존재해야만 모든 결정 문제를 해결할 수 있는데, 알고리즘은 $\aleph$개만 존재하며, 칸토어가 밝혀낸 대로 둘다 무한히 많은 수이지만 $2^{\aleph}$리 $\aleph$보다 크므로,

우리는 가능한 모든 결정 문제 각각에 대한 알고리즘을 항상 찾을 수 없다. 이로써 현대의 컴퓨터는 완전한 계산 기계로써 고안되었지만, 가능한 모든 계산을 할 수 있을 뿐 가능한 모든 문제를 풀 수는 없다.

소프트웨어가 어떤 알고리즘을 실행할 때 그 알고리즘이 올바른지 확인하는데도 어려움이 있다. 알고리즘이 올바르다는 것은 모든 입력값에 대해서 옳은 답이 나온다는 것이다. 그렇기 때문에 알고리즘의 입력값으로 가능한 것의 수가 유한하면 모든 각 입력값에 대해서 알맞은 답이 잘 나오는지 확인해 그 알고리즘의 정확성을 파악할 수 있다. 하지만, 앞서 말했듯이 모든 표기는 자연수로 나타낼 수 있어 가능한 입력값의 수도 $\aleph$개이다. 일초에 하나의 입력값을 확인한다 해보자. s초 동안 s번째 입력값까지는 모두 올바른 답이 나오는 것을 확인했어도 $s+1$번째 이후의 입력값에서 오류(error)가 날지는 s초가 지난 현재로써는 알 방법이 없으며, 이는 s가 몇이든 적용된다. 이를 해결해보고자 컴퓨터를 여러대 작동시켜도 마찬가지다. 1초에 입력값 하나씩을 확인하는 컴퓨터 c대를 사용한다고 하더라도 s초 동안 $c \cdot s$번째 입력값만 확인할 수 있으며 $c \cdot s+1$번째 이후의 입력값에 대해서는 알 수가 없다. 그러므로 아무리 무한 메모리와 무한개의 실행기가 주어진다고 해도 알고리즘의 입력값의 집합이 무한히 크면 그 프로그램의 정확성을 확실히 보이기는 어렵다.

힐베르트의 완벽한 수학을 향한 꿈이 무너졌듯이 완벽한 소프트웨어를 향한 꿈도 허황된 꿈이겠지만, 그래도 수학이 이 세상에서 활발히 사용되는 것처럼 현실에서 소프트웨어는 이런 한계점들을 고려를 하면 잘 활용할 수 있다. 현실적으로 고려를 한다는 것은 간단히 이런 것을 말한다. 수학적으로는 자릿수가 1억, 1조인 수가 존재할 수 있지만, 우리의 현실에서는 그런 수를 마주할 일도 활용할 일도 극히 드물다. 문제를 제한하거나 크기를 줄일 수 있는 조건이나 프로그램의 성질들(입력값에 대한 정보, 불변량(invariant) 등)을 잘 정의하고 수집할 수만 있다면, 자주 쓰이는 프로그램에 대해서는 적당히 올바른 알고리즘을 찾을 수 있을 것이다. 예를 들어, 항공기 예약 시스템에서 승객의 비행 횟수를 활용하는 알고리즘이 있다고 해보자. 이 수 또한 모든 자연수가 될 수 있다. 무한히 많지만, 현실적으로 제한을 할 수 있다. 1회 최소 비행 시간은 1시간이고, 지금까지 200년을 넘게 산 사람은 없으므로, 비행 횟수로 들어오는 최소 1부터 최대 $24 \cdot 365 \cdot 200 = 1,752,000$까지일 것이라고 추상화(abstract)할 수 있다. 이 알고리즘의 비행 횟수에 대한 정확성을 확인하고자 한다면, 1초에 입력값 하나씩 확인하면 $1,752,000$초, 약 487 시간이면 답을 전부 확인할 수 있다. 만약 우리가 정의한 조건에 들어맞지 않는 경우가 나중에 생긴다면 다시 확인이 필요하겠지만, 적어도 그 전까지는 이 방법을 활용할 수 있다. 이처럼 주어진 문제를 잘 알고, 활용할 소프트웨어에 대한 지식이 충분하다면, 해당 소프트웨어를 현실에서 원하는 목적대로 씀에 있어서 최대한 올바르게 작동시킬 수 있을 것이다.

사실 수학은 순수학문, 컴퓨터는 공학에서 다뤄 컴퓨터가 가지는 한계점은 순수학문과 공학 사이의 괴리에서 오는 것이라고 착각하기 쉽다. 실제로 그 괴리에서 오는 문제점들도 존재한다. 하지만 소프트웨어가 가지는 근본적인 한계인 '모든 문제에 대해 실행 가능한 옳은 알고리즘이 존재하지 않는다'는 점은 오히려 추상 학문인 수학 자체가 가지는 불완전성, 모순, 결정 불가능성으로부터 자연스럽게 따라온 것으로 볼 수 있다. 그렇다고 절망적인 것만은 아니다. 그 한계점을 최대한 극복해내기 위한 다양한 방향의 연구가 컴퓨터를 계속해서 발전시키고 있다. 그중 보안이 소프트웨어의 한계점을 역으로 활용하는 분야라고 생각한다. 아무리 커다란 발전이 일어난다고 해도 한계점의 가장 단단한 핵은 깨지 못하겠지만, 핵을 둘러싸고 있는 껍질이 어떻게 벗겨질지 기대가 된다.